AD-A063 303 NAVAL RESEARCH LAB WASHINGTON D C F/G 9/2
REMOS: A REMOTE COMMUNICATIONS OPERATING SYSTEM FOR REAL-TIME A--ETC(U)
DEC 78 S SUTTON

UNCLASSIFIED NRL-8275 NL

/ OF /

AD
A063303

END
DATE
FILMED
3--79
DDC

LEVEL II

NRL Report 8275

REMOS: A Remote Communications Operating System for Real-Time Activities

STEPHEN SUTTON

Mechanics of Materials Branch
Ocean Technology Division

NRL-8275

Final rept.

20 December 1978

21 p.

RR 023 03

NAVAL RESEARCH LABORATORY
Washington, D.C.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NRL Report 8275	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) REMOS: A REMOTE COMMUNICATIONS OPERATING SYSTEM FOR REAL-TIME ACTIVITIES		5. TYPE OF REPORT & PERIOD COVERED Final report on NRL Problem 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Stephen Sutton		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Research Laboratory Washington, D. C. 20375		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NRL Problem F01-04 Project RR023-03 — 61153N
11. CONTROLLING OFFICE NAME AND ADDRESS Department of the Navy Office of Naval Research Arlington, Virginia 22217		12. REPORT DATE December 20, 1978 13. NUMBER OF PAGES 20
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer network
Data acquisition
Distributed processing
Materials testing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

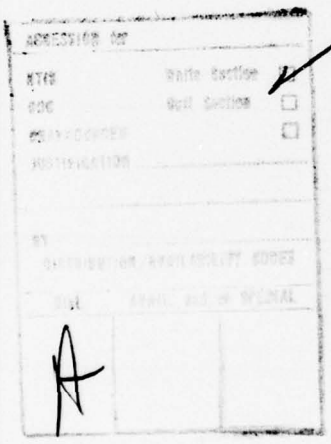
REMOS is a software system that allows a host computer, usually a large time-shared computer, to serve as a background operating system for a smaller satellite dedicated to some real-time activity such as data acquisition or process control. REMOS allows the bulk of the processing associated with an entire real-time activity to be performed as background processing in the host, where a wide variety of services and utilities is available for data processing, storage, display, etc. The system also can, through the satellite computers, be dedicated as needed for many real-time applications.

(continues)

69 01 10 [illegible]

20. ABSTRACT (Continued)

The REMOS software was designed for a high degree of host transferability, so that a minimal investment in the satellite and access to any large time-shared system gives the user a versatile real-time system. REMOS is an open-ended system whose goal is to provide a fundamentally sound framework upon which more-complicated, specialized systems can be constructed.



CONTENTS

INTRODUCTION 1

THE REMOS SYSTEM 2

General 2
Design Techniques 3

PROCESS STRUCTURE AND DISTRIBUTION 3

General Configuration 3
Communication Link 5

PRIMITIVE MODULES 5

Line Handlers 6
Kernel-Control Modules (KCM's) 6
Process Synchronization 7
The Exchange-Synchronization Primitives 7
Multiple-Satellite Addressing 8

MIDLEVEL CONCEPTS 9

Process Segmentation 9
Process Control Segment (PCS) 10
Access by Segment 10
Standard Rendezvous 10

HIGH-LEVEL CONCEPTS 11

Process Migration 11
Context Switching 12
Run-time Remote Support 12

SATELLITE PROGRAM CREATION 13

USER INTERACTION 14

The Stream Demon 14

SUMMARY 16

REFERENCES 17

REMOS: A REMOTE COMMUNICATIONS OPERATING SYSTEM FOR REAL-TIME ACTIVITIES

INTRODUCTION

The digital computer has been a revolutionary and powerful tool for the solution of scientific problems from its very inception. Although most of this power and versatility has been directed toward non-real-time analysis, the digital computer is beginning to have an influence on the real-time collection of empirical data. Ultimately, the joining of these capabilities will provide immense power for the solution of complicated and previously intractable problems involving the complex analysis of large amounts of experimental data.

If this goal is to be fully realized, a wide variety of computational resources must be utilized. Minicomputers or microcomputers offer a cost-effective approach to the implementation of local real-time nodes in a larger process, but they suffer from a lack of computational power for larger programs. Large-mainframe computers offer a cost-effective answer to the solution of large analysis problems, but they are generally too expensive to be used as real-time controllers.

A very attractive solution is a union between these two types of computers, in which a network of small, inexpensive "satellite" minicomputers or microcomputers is under the general control of a much larger mainframe or "host." This allows the data collected by the satellites to be immediately available to the host on a pseudo-real-time basis (that is, the host will not have to respond instantly to real-time events, but it will respond "soon," on the order of normal time-shared delays) so that large and sophisticated analysis programs can evaluate the data and, presumably, alter the control algorithms in the satellite.

Typically, an installation that has access to any large-mainframe computer equipped with a wide variety of analysis and applications packages would be linked to a small, portable, inexpensive computer that would be located in proximity to a data-acquisition activity. The satellite, under the general control of the host, could then be dedicated only to the task of gathering the data and controlling the testing apparatus, and it would transfer the data back to the host for analysis. The user in this system would normally remain in direct contact with the host computer, which has the broad range of facilities he needs, and the satellite could simply be viewed as a real-time extension of the host.

There are many practical constraints to the design of such a system. In many cases the satellite computers require much supporting hardware, such as disk drives, to support large, local operating systems. These devices can make satellites too expensive to be used at many locations and can increase maintenance costs and down time.

It is also apparent that software costs are a major expense in the creation of small computer systems, and they are a major consideration in the design of such systems. The extremely desirable goal of implementation-independent software can be realized by the

Manuscript submitted August 31, 1978.

coding of as much of the system as possible in a higher-level language by the use of standard programming constructs and good programming techniques such as modularization. The mainframe computer, with its wide variety of programming aids, is generally best equipped to handle this task in a cost-effective manner.

REMOS (REMote Operating System) was designed to accommodate these constraints and is an attempt to design a viable, cost-effective system for linking real-time control to mainframe computing power in a relatively hardware-independent environment. This paper documents the various design features and the use of the REMOS system, and its content should be of interest to those engineers and scientists who are automating their data acquisition facilities, as well as to the designers of distributed and real-time computer systems.

The contents of this paper should also serve as a brief tutorial in the concepts of distributed computation, since many of the ideas and methods common to this area are addressed in the REMOS system. Of particular interest is the process-synchronization mechanism, since REMOS is probably the first operation system to implement this relatively new technique.

THE REMOS SYSTEM

General

REMOS (REMote Operating System) is a software system designed to fuse the resources of any large time-shared computer (host) and a small, inexpensive minicomputer or microcomputer (satellite). The REMOS system was designed for situations in which one or more satellites are dedicated to some real-time function such as data acquisition or process control, but it is flexible and versatile enough to be used in a variety of applications.

In REMOS the host computer serves as the operating system and controller for the satellite computer. The REMOS design philosophy is that every bit of background or non-real-time processing is brought into the host computer, where a wide range of programming resources can be effectively utilized. The satellite performs only the real-time activities and can therefore be small, inexpensive, and portable, with no large peripherals. The entire system remains under the control of the host computer.

The REMOS system was designed for a high degree of transferability between host computers; this transferability was accomplished primarily by the programming of the host-resident REMOS modules in a high-level language and careful separation of any computer-dependent modules. The particular satellite chosen for the REMOS system was the LSI-11, a minicomputer made by the Digital Equipment Company.

One of the primary design goals of the REMOS system is that it be an open-ended system that can be easily adapted to a variety of applications. REMOS provides the host computer with a very powerful and versatile set of control primitives with which to control the satellite computer. They are described in the section "Primitive Modules." Added to these primitive capabilities is a flexible set of operations for transporting satellite programs and data between host and satellite; these are described in the "Midlevel Concepts" section. The combination of primitive and midlevel capabilities forms the basis for the most sophisticated functions of the REMOS system, and these are described in the "High-Level Concepts" section.

The remaining sections of the paper present techniques for creating satellite programs on the host computer and some practical tools to be used to attach an interactive user to the system.

References 1 and 2 describe a data-acquisition system that uses the REMOS system and may be of particular interest to those involved in materials testing.

Design Techniques

The design techniques employed in the development of REMOS reflect the general philosophy behind the system. The stated goal was to give prime consideration to system portability, reliability, durability, and efficiency. These are in a sense all interrelated, and lead ultimately to a system that is easy to maintain, modify, and transport between uses. These are precisely the goals, techniques, and methods of the structured programming used in the REMOS system.

The REMOS system was designed by use of a top-down approach and was implemented in modular, self-documenting code. Flexibility to adapt to different host and communication lines was built into the system design to ease translation to a new host. At present, REMOS services only the LSI-11 satellite, but adaptation to any of the current microcomputers or small minicomputers would not require a major rewriting of the system except for the cross-compilation facilities.

PROCESS STRUCTURE AND DISTRIBUTION

General Configuration

In the discussion below, a single satellite will generally be assumed to be under the control of the host-resident operating system (Fig. 1). Multiple satellites can be controlled by a separate REMOS module in the host dedicated to each satellite or by a single REMOS host package controlling several satellites (shown in brackets in Fig. 1). REMOS has facilities to allow this to be easily accomplished and these are discussed later in the paper.

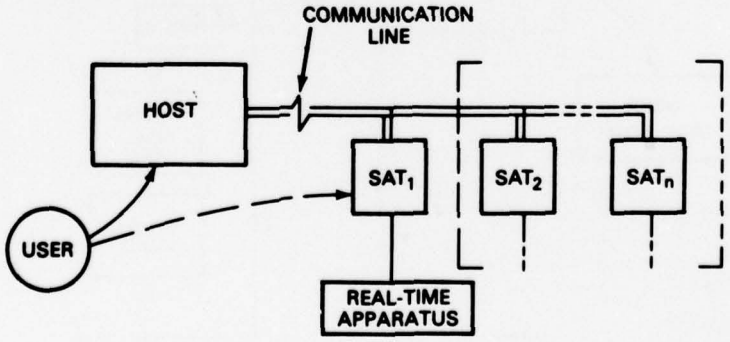


Fig. 1 — Host-satellite configuration

There are distinct roles played by the various processes in the system. The term "process" has come to have a distinct but vaguely defined connotation in the field of operating-system and interprocess communication theory. In this paper one can assume that a process is simply a computational entity that performs some function. It may be a single software program, or a small set of programs working together toward a common purpose.

The first realm to be considered is that of the "user" processes (Fig. 2). These generally change from application to application and are concerned with accomplishing the overall data acquisition task. Each such user process can be logically separated into a real-time portion and a background portion. The real-time portion is composed of those activities that must be responsive to some external events and will not function properly if that response cannot be maintained. The remaining activities of the user process are classified as background activities. The general philosophy of the REMOS system is to perform all background activities in the host, programmed as one would normally program the host, and relegate only the real-time activities to the satellite. REMOS provides the necessary facilities for transferring data between the background and real-time portions of the user process.

The second realm to be considered is that of processes in the satellite that act in support of the user process and do not in general change from one user activity to another. These processes are generally called the operating system of the satellite and would traditionally be placed within the satellite. The REMOS system, however, puts those operating system activities that are not real-time in the host computer. The operating system functions that must provide real-time support of the user process are simply attached to the satellite real-time process as subroutines.

REMOS is thus almost entirely in the host, and the host assumes all the duties and powers of an operating system for the satellite except that it manages the satellite remotely across a communication link. REMOS allows the background user processes in the

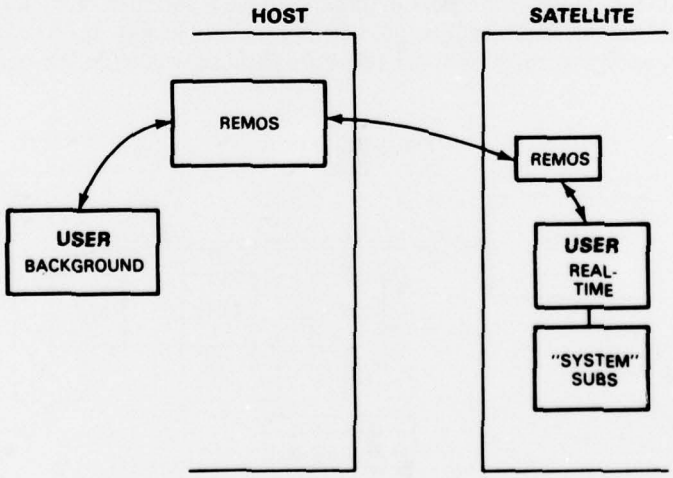


Fig. 2 — Logical structure of a user process

host to communicate easily with the satellite user process. REMOS was designed to allow the user to extend the operating-system services without a detailed knowledge of the internal performance of the system. REMOS is actually a little broader than a normal operating system in that it provides such services as cross compilation, satellite process loading and unloading, and various forms of data transmission between the host and satellite user processes.

There are several advantages in this configuration. The operating system is now contained almost entirely within the host and can be programmed in a high-level language. The system then becomes highly transferable between host computers, and it is easy to maintain, update, and modify. All implementation-dependent processes in REMOS are carefully separated for ease in translation to a new system. This interchangeability provides security from host failure, durability (since languages tend to last longer than computers), and the convenience of choice of the host that currently provides the greatest economy and power for a particular job. Although the use of a high-level language itself is not enough to guarantee this, the use of structured-design technique and modularity within that language does.

REMOS is programmed in ANSII standard FORTRAN chosen not particularly for convenience but for interchangeability. This enables the features of the system to be easily tailored or extended to a particular use.

Communication Link

REMOS is independent of the particular type of communication link that connects the host and satellite(s). For example, this link can be a high-speed, parallel, direct link between the two, or a serial, asynchronous or synchronous data link that can be transmitted across telephone lines. The primary usage at the Naval Research Laboratory is of the serial, asynchronous links that allow the satellites, which are physically portable, to be transported to remote sites in the field and to be controlled via phone line.

Actually, any existing communication link, however complicated, could be used to connect the two, but high communication overheads run contrary to the design principals of the system.

The details of the communication line are not of particular interest here, since this tends to be somewhat implementation dependent. Instead, the functionality of the host's remote management and servicing of the satellite user program will be of prime interest.

PRIMITIVE MODULES

Figure 3 shows a diagram of the general REMOS system configuration. The system has several hierarchial layers of processes, and processes at higher levels depend upon those below to service their requests. At the lowest level are the primitive modules that provide the host a fundamental control over the satellite. The user has access to each level of REMOS, but the upper levels tend to do larger tasks in a more automatic fashion. Access to the low-level modules provides the user with the ability to customize the REMOS system.

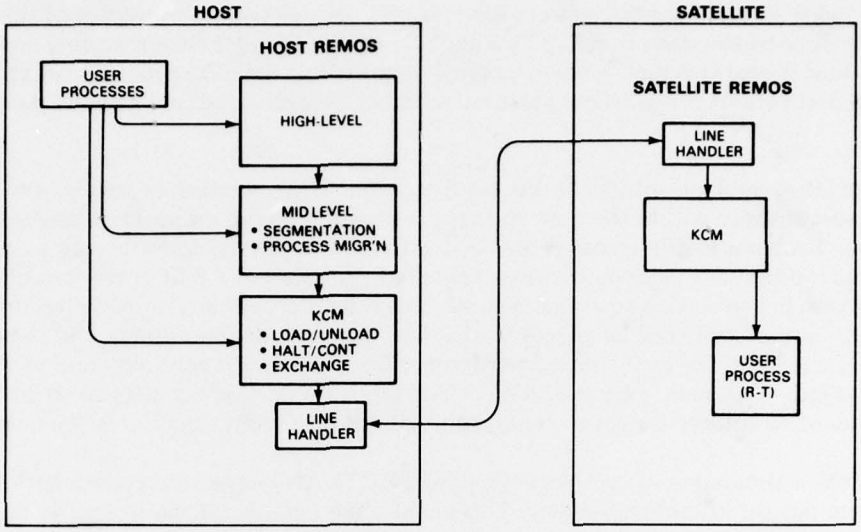


Fig. 3 — REMOS logical structure

Line Handlers

There are two processes termed the line handlers, one in the host and one in the satellite, that handle the transmission of raw data across the communication link.

The basic responsibility of the line handlers is to assure that the correct information is passed between the kernel communication modules. The line handlers have the responsibility for line-error detection and correction and perform any necessary buffering or translation of the incoming line information so that it is presentable to the appropriate kernel module. The line demons are designed so that they are the only modules in REMOS that depend upon and will change with the actual line type used. In addition, the host line demon and file handlers are the only modules in the host portion of REMOS that will vary with host type. This rigid separation of the machine- and line-dependent modules provides for excellent transferability of the system.

Kernel-Control Modules (KCM's)

The Kernel-Control Modules (KCM's) are those modules that give the host fundamental control over the satellite. There are basically three such capabilities needed for REMOS: The host must be able to write anywhere into the satellite's memory, to read from the memory, and to suspend (and resume) the current user program.

The "load" kernel module can load any contiguous block of information into the satellite's memory, and the "unload" module can read any such block. The "halt" and

"continue" modules control execution of the user process in the satellite. These kernel processes in the satellite are interrupt-driven and have priority. This allows the host complete control over the state of the satellite and precludes a lockout by a rampant user process.

All three of these capabilities are controlled by the host in a strict master-slave fashion. The satellite cannot forcibly load or unload its memory but can only indicate a request for such actions, which must eventually be serviced by the host KCM's.

Process Synchronization

Processes in the host and those in the satellite (whether operating system or user) are asynchronous processes and require a means for process synchronization. By this we mean simply that these processes essentially run at their own independent pace. If two processes must exchange information, one may have to wait for the other to reach that place in its execution sequence where it will be ready to exchange the information. There must in effect be a "meeting of the minds" between the two processes; synchronization primitives are the low-level mechanisms that allow the processes to synchronize. This synchronization is a necessary mechanism for operating systems in general, and is an important mechanism in the REMOS system.

An example of the need for synchronization is the case when the real-time satellite user process has completed a portion of its duties and needs to inform the host, which is checking periodically as its time-shared workload permits. There are many other examples of the need for this synchronization that are more intimately related to the internal duties and functions of the REMOS operating system.

The history of interprocess synchronization primitives began in the early days of computers with the test/set primitives, then proceeded through the very powerful semaphors of Dijkstra, and then to such concepts as ports and mailboxes. Very recently, Zave and Fitzwater [3] have taken a nicely functional approach to this problem with the introduction of the "exchange" synchronization, which was chosen for the REMOS system for its power and flexibility.

The Exchange-Synchronization Primitives

Consider two independent processes A and B that wish to communicate by passing some information to each other. This is shown schematically in Fig. 4, where the encircled numbers indicate the time sequence of the following description. They are allowed to exchange packets of information through a particular addressable (named) data-collection point called a "rendezvous." This exchange must be performed at a time when both are ready to do so (a "meeting of the minds"). When either wishes to perform the exchange, it will call an exchange function to perform the exchange. This function, "exchange to communicate," we will abbreviate by "XC." The XC function receives two arguments from its caller, a rendezvous site, and an argument to be passed to the other process through the rendezvous site. The XC function then presents this value at the rendezvous site (step 1) until the corresponding XC function for the other process (B) presents its value (step 2) by calling the XC function for the same rendezvous site. At this time an exchange of values is performed (step 3),

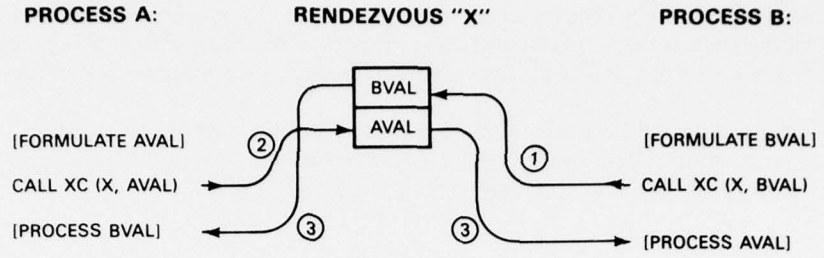


Fig. 4 — Synchronization exchange functions

and each XC function returns this value to its caller (i.e., A's value goes to B, and vice versa) and the callers continue on their own ways with their newly exchanged values (step 4).

The values that are exchanged, in general, have some agreed-upon meaning between the two processes. From a functional point of view, as expressed by Zave and Fitzwater, these values may be arbitrarily complex data structures. The REMOS processes may pass values which are self-contained information or which may be pointers to large tables of information.

There is a second exchange primitive called "exchange to synchronize" (XS) which is identical to the XC function except that when a value is presented to a rendezvous site, if there is no corresponding value with which to be exchanged, the function will return and report a failure. Two processes may synchronize by both using the XC or by one using the XC and the other using the XS, but both may not use the XS.

In the REMOS implementation, these functions need to be modified slightly to attach a time-out value to the XC primitive when the functions are executed by the host. The XC so modified will return the null value if an exchange is not made within the time-out limit. With this addition, the XS simply becomes the XC with a zero time out.

The power and utility of these primitives are probably not yet obvious at this point unless the reader has had previous experience in this field. Reference 2 has a more thorough treatment of this subject. These synchronization primitives perform a necessary function for many of the high-level constructs in REMOS that are described below.

Multiple-Satellite Addressing

The REMOS code is structured to allow one host to control several serially-connected satellites as illustrated in Fig. 1. This requires the inclusion of the Stream Demons in the satellite. (The Stream Demons are discussed thoroughly in a later section.) This is accomplished by making all REMOS service modules in the host reentrant, so that each request from the user to REMOS must specify a data base that characterizes the particular satellite to be accessed. This facility allows many satellites' activities to be synchronized, perhaps to accomplish some common real-time task.

MIDLEVEL CONCEPTS

The next level in the REMOS process hierarchy is that of the midlevel concepts. These are host-system independent functions that have widespread use in the manipulation of the user satellite programs and would be used frequently in any extension or modification of the REMOS system.

Process Segmentation

A logical structure must be imposed upon the user process so that it can be effectively managed. Each satellite user process is created as a set of logical segments. A logical segment is a contiguous section of the satellite's memory that may contain either executable code, or data, or both, or may contain no usable information. These segments then become the units for transferring processes or process parts through the system. In many cases these bear a strong resemblance to the usual type of segmentation used by many large computers, except that the division is a logical one and of no use to the satellite hardware.

Figure 5 displays the logical segmentation of a typical user satellite process. The general logical structure remains intact, whether the process is loaded into the satellite or is stored in the host's filing system.

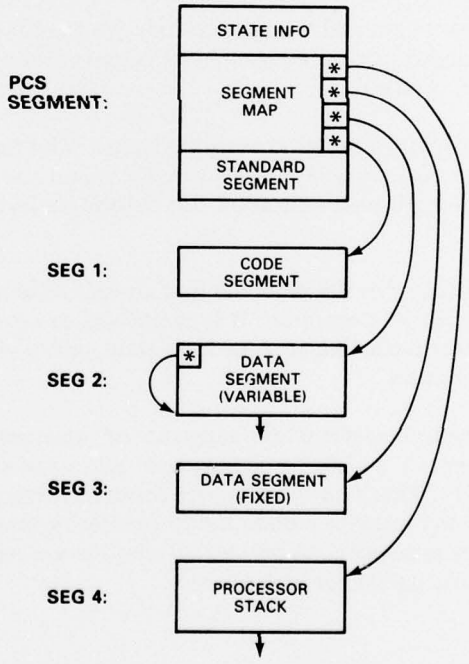


Fig. 5 — User satellite-process segmentation

Process Control Segment (PCS)

The first of these segments is called the Process Control Segment (PCS) and is used to store information about the process. All the information necessary to restore the state of the processor is kept within the PCS, i.e., the general purpose registers, the program counter register, and processor status information. This "state" information allows the process to be halted, removed from the satellite, and later restored and restarted without logical interruption. The PCS also contains a complete map of the location and length of each segment that is a part of the process, including itself. The PCS is managed by the REMOS operating system and need be of no concern to the casual user; however, it is easy for the user to access the information in the PCS for any special purpose.

The REMOS PCS is analogous to the technique used by most multiprocess operating systems to perform these functions.

Access by Segment

There are few rigorous rules for segmenting a user process, but a process is usually segmented according to how it is to be accessed. Segments may be either of fixed size or of variable size and they may be either absolute or relocatable. Absolute segments will not function properly if loaded into any other section of memory. Relocatable segments can be relocated in memory without being changed.

A code segment contains executable machine code for the satellite and is of fixed length. Generally these segments are only transferred between host and satellite when the user process is being loaded or saved.

Data segments can be of fixed length or varying length. The first word of a variable-length segment contains the current length of that segment and must be kept correct by the satellite program. Variable-length segments allow the automatic transfer of the minimal amount of data necessary.

The processor stack segment is the segment that contains the program working stack that is used heavily by the LSI-11 Computer. It is important to save this stack, when a satellite program is suspended, for restarting at some later time, and this is done automatically by the REMOS midlevel modules.

REMOS has the ability to transfer whole segments of information by their logical designation (i.e., "third segment") without its having to know the exact location of the segment in the satellite memory. This greatly simplifies those operating-system processes that must manipulate segments and helps to isolate the lower-level system details from the higher level. Moreover, user service processes can be created that have widespread application and isolation from the details of a particular user process.

Standard Rendezvous

The PCS also contains a set of standard rendezvous locations for use by the exchange synchronization functions, the number of which may vary from one user process to another.

These are available for use by either host or satellite in any mutually agreed upon fashion. The advantage of including a standard set of rendezvous is that they can be accessed by logical addresses instead of by their actual memory addresses in the satellite. This aids in the creation of operating-system routines to service the user process during run time (we will describe this in more detail later) since these routines can synchronize through these standard rendezvous locations.

HIGH-LEVEL CONCEPTS

The final realm of REMOS functions we consider implements the high-level constructs of the system. These are in some sense the least-precisely defined functions, since they are the ones that are most likely to be tailored to a particular application. This is the level at which the system will do most of its "growing," where the user who wishes to tailor the system to a particular specialized environment will make additions. The ideas discussed below are intended to display the kinds of situations where the REMOS system could be applied, and most of the concepts are in various stages of development at the Naval Research Laboratory.

Process Migration

Entire user processes migrate through the REMOS system, the principal path being between the host filing system and the satellite, as illustrated in Fig. 6. REMOS has standard loading and unloading modules that can be requested to load into the satellite an entire user satellite process from the host's filing system or unload one from the satellite into the host's filing system. A loaded process may either be one that was previously suspended and brought back to the host or a fresh one being loaded into the satellite for the first time.

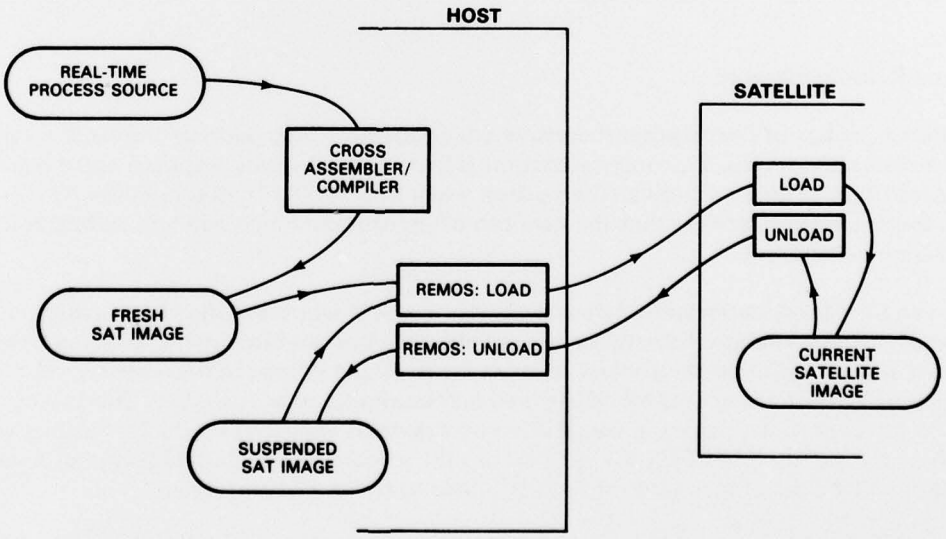


Fig. 6 — User-process migration

Context Switching

More than one process can be active within the satellite by means of context switching. Context switching is employed by operating systems to allow several user processes to use a single computer with each not aware of the sharing at all. One method used is to keep several active user processes in the satellite and time slice, the satellite CPU among them, either under control of the host, or under local control of a scheduler within the satellite. A second method used is to allow the host to maintain a number of active user processes in its files and swap one at a time into the satellite so that it can have the entire satellite to itself for a period of time called its "time slice." Although the transmission of entire processes causes high overheads, this approach may work quite well for low-duty-cycle activities. For fast context switching, the processes will have to be kept in the satellite and scheduled by a satellite resident scheduler.

Although the original design goal was to dedicate the satellite to one data activity, it can easily be extended to a multitask environment when necessary. An important point is that the control essentially remains within the domain of the host processor.

This multitask configuration is used because it is better to keep such a set of logically independent tasks separated physically than to create one large task to perform the unrelated duties of the separate smaller tasks.

This context-switching scheme can be further modified to provide practical multiprogramming facilities. If the satellite is to be dedicated to a single class of real-time apparatus, for example, several identical testing machines performing similar tests, then the user satellite process can be made reentrant and a different data base can be used for each separate real-time activity. This provides essentially as many logical processes as there are data bases. The data bases may, of course, be either memory resident or swapped as time considerations allow. Also, the synchronization primitives provide the means to implement these types of multiprocessing systems.

Run-time Remote Support

There is a class of direct-support services which the host may perform during the execution of a real-time process. The only restriction is that the host is not real-time and cannot be counted upon to give its undivided attention when it is wanted by the satellite. Nevertheless, there are some services that the host can often provide which will not unduly restrict the real-time capabilities.

Among the most important of these real-time support services is the virtualization of the memory of the satellite. Effectively, we are allowing the satellite process to use a larger addressing space than it has in physical memory by shuffling groups of data between the satellite and the host under control of the host and unknown to the satellite. This is very similar to the concept of paging in paged computer systems except that address calculation is not done by the satellite CPU (unless of course the satellite has this capability) but must be performed by a set of standard routines available to the satellite program.

One form of memory virtualization called "treadmilling" is a system for dumping raw data to the host while the user satellite process is active. Essentially, the satellite places its

data on a circular queue, perhaps in logical blocks of some specified length, and the host works as fast as it can to unload these blocks from the satellite; a common technique is to store them on a file in the host. Of course, the satellite cannot load the queue faster than the host can empty it on the average, although short filling bursts are possible. This technique works well for fairly low duty cycle processes that run for a long time (perhaps days) and generate a lot of data. The data can be processed by the host as it is received, or it can be simply stored until the data acquisition activity is completed.

It should again be noted that these types of services will rely heavily upon the synchronization primitives.

SATELLITE PROGRAM CREATION

One of the most important services of the host that does not directly involve real-time support is satellite program creation. It is highly desirable that the host be able to cross-assemble or cross-compile programs for the satellite. Satellite program creation is entirely a background activity and the host is ideally suited to this type of activity.

Since the satellite's duties are primarily real-time, languages that are efficient both in space and in execution time are desirable, yet a structured language is desirable to ease the pain of low-level programming.

REMOS will incorporate two types of program-development facilities. The first and most basic of these is a cross-assembler to assemble standard assembly language for the PDP-11 instruction set for the LSI-11. The binary output of the assembler will be in a format directly accessible to the REMOS load module.

Linking, the process of gathering together the subroutines needed to complete the user satellite process, will be done entirely at source level in the REMOS system. That is, separately assembled binary subprograms will not be linked to form a single executable process, or "load module," but the source codes will be linked before assembly, although several distinct program entities, each with its own name scope, can be assembled at once. The primary advantage of this process is that the procedure for linking and loading is greatly simplified, and the troublesome distinction between binary and source libraries and their separate access routines is eliminated. The only price paid for these conveniences is a slower compilation time, but since compilation should not be frequent this is not a great burden.

The second facility for program creation that will be designed and implemented is a high-level, structured programming language. The language will implement the fundamental structured concepts yet retain good run-time efficiency.

Figure 6 illustrates the flow of satellite user processes through the system. Note the direction of flow of the process images and that suspended images can migrate to and from the host under the control of REMOS.

Both the cross-assembler and cross-compiler will be written in the same language as REMOS (FORTRAN) to continue the host-interchangeability feature.

USER INTERACTION

It is important to consider the role played by the user as an interactant in a system such as the REMOS system. Communication between the user process in the satellite and an operator at an interactive terminal is a common requirement for real-time systems. One means of achieving this communication is to allow the satellite to use its own I/O ports to interrogate the operation directly.

An alternate and better technique is to let the host handle all operator communication, perhaps at the request of the satellite, and pass succinctly coded results to the satellite. The host is more-readily equipped to handle operator communication, since this involves a lot of string parsing and decoding, and for a real-time system probably it should employ some error checking. This activity is an unnecessary burden on the satellite that is better avoided if possible. Again, we are transferring any non-real-time processing to the host computer, where it is most efficiently done.

The Stream Demon

A typical real-time or data-acquisition operational setup is shown in Fig. 7. The remote satellite is at an experimental site, perhaps a laboratory, and the host is a large time-shared computer removed from that site. There are two means to attach the user to the system in this simple configuration. The first is shown as the A path in Fig. 7, with user and satellite connected to the host by different communication lines or paths. The second technique (path B) has the user and satellite both share the communication line with the host; it is of most value when the satellite and user are both removed from the host and the communication path is probably a telephone line.

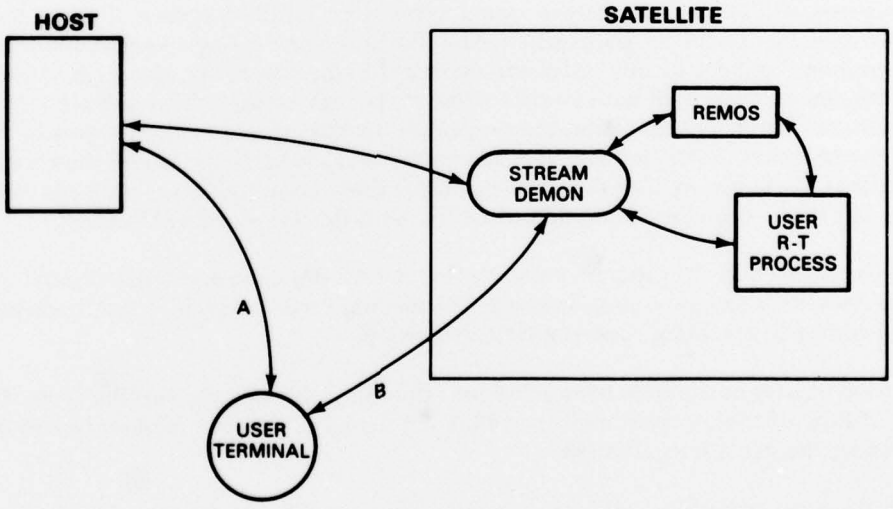


Fig. 7 — User interaction in the REMOS system

In the latter configuration, there have to be three logically distinct communication paths. The user must be able to communicate with the host just as though he were a normal time-shared user; the host and satellite must be able to pass their REMOS-directed messages between the line handlers; and finally, the user and satellite should be afforded a communication path for times when the host must be removed from the system.

To allow REMOS to be used in this manner, a small module called the "Stream Demon" was incorporated into the satellite between the host and the satellite's line handler. All of the communication paths are logically connected to the Stream Demon through a "port," and the Stream Demon provides for the proper routing of the messages among these paths (ports). The logical structure of the Stream Demon is illustrated in Fig. 8.

The Stream Demon, it should be emphasized, is a software module, although its actions are very much like a hardware switch. The Stream Demon has several logical, bidirectional streams of information connected to it, usually character streams. These could be from a user terminal or from the interface to the remote host, or they could be considered to originate from a process within the satellite. At any given moment, the Stream Demon contains an interconnect map that determines for each input stream a list of output streams to which the information is to be passed. The interconnect map can be changed at any time by any of the external sources by its sending a message specially encoded for the Stream Demon.

Figure 8 shows the interconnect map schematically. There are three distinct paths defined, I, J, and K. Consider the situation in which J is the only active path. This corresponds to the user at his terminal communicating with the host computer in the normal time-sharing mode with the satellite not a party to the communication. The user might at that time initiate a program on the host that calls a REMOS function for information transfer to or from the satellite. The host-resident REMOS then sends a coded message to the Stream Demon to

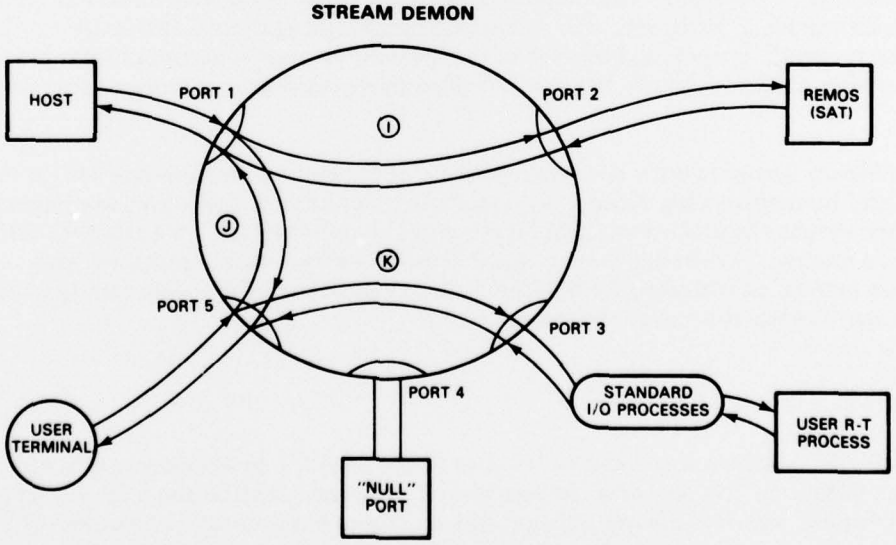


Fig. 8 — The Stream Demon

reconfigure the interconnect map so that path I is the only defined path through the Stream Demon. REMOS then sends or receives the information to or from the satellite and, when finished, reconfigures the interconnect map so that path J is the only active path, and the user is back in business.

Path K is the one to be used if the host is off-line for a period and the satellite needs operator interaction. Note that it is possible for multiple Stream Demon paths to be active. It is therefore possible for the satellite to be left running at the same time the user is using the host (path J) for background or program development. This allows the user to use the system in a time-sharing mode without interrupting ongoing real-time activities, yet the ability to halt or interrupt the satellite through REMOS is maintained.

The ability to redefine the interconnect map is on a priority basis, and each request to reconfigure has a priority associated with it. These priorities are not built into the system but can be flexibly assigned, higher priorities usually going to the more reliable or control-type processes. The highest priority is reserved for the user terminal to allow the user to stop any runaway condition.

A single host can be configured to control a bank of satellites across a single communication line with the host using the Stream Demons to route a particular message to the correct destination satellite, as in Fig. 1. The single line can be connected in parallel to all the satellites, and the host then addresses a particular satellite by sending the same interconnect prompt to each Stream Demon. The particular interconnect command is chosen so that all satellites except the one being addressed route the host message stream to a "null" port. Subsequent characters coming across the line will enter all Stream Demons simultaneously, but in only one satellite will the stream of characters be sent to the REMOS satellite-resident line handler. In all others, they will be routed to the "null" port. The addressed satellite stays active for information transferral until the host redefines the interconnect maps.

In addition, the Stream Demon houses a standard set of I/O service calls (Fig. 8) which allow satellite resident processes, such as the user process, to exchange information with the user at the terminal. They would be used in cases where the host is detached from the system and are exactly analogous to the I/O service-call rotines provided by a resident operating system.

The Stream Demon allows the simple sharing of the communication line among the host, satellite, and interactive user. From a very practical point of view this allows the implementor of real-time systems to make a minimal investment in equipment (i.e., the satellite) and, with the aid of a remote time-sharing system, gain access to a very versatile real-time data-acquisition system. In addition, the Stream Demons can be used for a wide variety of message-handling and message-routing applications.

SUMMARY

The REMOS system was designed because of the need for an effective means of combining computational resources for the solution of complex scientific and engineering problems that require real-time data acquisition and control of experimental apparatus. The system allows a large, time-shared-mainframe host computer to control smaller satellite

computers that are in turn dedicated to the data-acquisition activity. Primary goals of the REMOS system are for it to be transferable between host computers, to be open-ended in the sense that its primitive capabilities can be recombined and augmented to build a wide variety of specialized systems, and to allow the use of inexpensive, truly dedicated satellite computers.

The structure of the REMOS system can be viewed as a hierarchical structure, which begins on the lowest levels with the simple communication handlers and controllers whose only job is to make sure that primitive information is correctly transferred between host and satellite. Any system-dependent I/O specifics are confined to these levels, which eases the job of transferring to a new host computer. Also on this lowest level are the very important means for interprocess synchronization which use the "exchange" communication primitive recently developed by Zave and Fitzwater [3].

The next level, or midlevel, is concerned with the segmentation of logical processes and the migration of these segments through the system. A variety of segments is available for the transfer of data or code to and from the satellite. All the control (and therefore all the code) for control of these midlevel and upper-level services is contained in the host and is coded into a high-level language for ease of maintenance, modification, and transferability.

The final hierarchical level is the highest and in some sense the least-well defined, since it is at this level that the system most rapidly changes and grows. On this level, many services can be developed for specialized activities such as multiprocessing and extension of the memory capabilities of the satellite.

A module called the Stream Demon was incorporated into the system in order to allow the user a convenient and flexible interface to the system. The Stream Demon has a general message-stream switching capability used in a variety of ways to facilitate the transmission of messages, primarily character-oriented message streams, through the system.

The REMOS system offers a practical, cost-effective solution to the problem of bringing a variety of computational resources to bear upon the problem of laboratory data acquisition, and it is currently incorporated into various materials-testing activities at the Naval Research Laboratory.

REFERENCES

1. S.A. Sutton, "A Model for Computer-Based Data Acquisition and Control," *Exp. Mech.* 17:141-146 (Apr. 1977).
2. S.A. Sutton, "Fracture Toughness of Stretched Acrylic Plastic," *J. Test. Evaluation* 6, No. 6 (Nov. 1978).
3. P. Zave and D.R. Fitzwater, "Specification of Asynchronous Interactions Using Primitive Functions," submitted to *IEEE Trans. Software Eng.*